Anime/Manga Recommendation System with IR Algorithm

December 2023

Team Members

Kanishaka Pranjal(S20210010108)

Swastik Mukati(S20210010221)

Description

This Project serves as an information retrieval system for Anime/Manga recommendations, integrating features that enable users to input their preferences and receive personalized suggestions. The system leverages an Information Retrieval (IR) similarity algorithm to match user preferences with Anime/Manga in the dataset. Recommendations are generated based on various factors, including anime/manga descriptions, ratings, user interactions, and the user's viewing history.

Built with Flask, the application also incorporates functionalities such as document search and ranking, spell correction for user queries, and a user feedback system. Users can explore and interact with the system through a web-based interface.

This integration offers a comprehensive platform for Anime/Manga enthusiasts, providing a seamless experience for discovering new content tailored to their tastes. Whether users are seeking recommendations based on descriptions, ratings, or personal viewing history, the system aims to enhance the overall Anime/Manga discovery process.

Tasks / Components

1. DataSet - Collected from My Anime List dataset from Kaggle

Main.csv

- This CSV file contains a list of 87,316 anime/manga entries.
- Path: main dataset/main.csv
- Columns: ID, Title, Description, Rating, Feedback, Evaluation, Normalized Evaluation

User_profile.csv

- This CSV file contains a list of 75,989 user profiles.
- Path: main dataset/user_profiles.csv
- Columns: User ID, Profile, Watch or Read

2. PreProcessing

Lowercasing:

- Converts all text to lowercase to ensure consistency and avoid case-sensitive discrepancies.

Tokenization:

- Breaks down sentences or paragraphs into individual words or tokens, essential for text-based analysis.

Stopwords Removal:

- Removes common words (stopwords) that do not contribute significantly to the meaning of the text, reducing noise in the data.

Stemming:

- Reduces words to their base or root form, grouping together words derived from the same root, even with different inflections or suffixes.

3. Indexing

Index File Format:

- Path: index.txt
- The index file is a .txt file storing term information in the format: term - (doc_id, term_frequency).

Indexer Function:

- The makeIndex() function is crucial for indexing anime/manga descriptions, processing each description and creating entries in the index file.

Index Loading:

- At runtime, the program loads the index.txt file to access the pre-built index.

makeIndexMap() Function:

- The makeIndexMap() function generates a dictionary containing term, doc_ID, and frequency without reading the entire document.

Indexing Workflow:

- **Term Extraction :** The makeIndex() function processes the description of each anime/manga to extract terms.
- **Index File Population :** For each term found, an entry is added to the index file in the format: term - (doc_id, term_frequency).
- **Efficient Data Representation:**
    - The makeIndexMap() function efficiently structures the indexed data into a dictionary for easy access.
    - This allows for quick retrieval of information related to a specific term, doc_ID, and its frequency.
- **Dynamic Index Updates :** As new anime/manga descriptions are added or existing ones are modified, the indexing process dynamically updates the index file.

- **Query Optimization :** The indexed data enhances search efficiency by providing a quick lookup mechanism, reducing the need to parse the entire dataset for every query.
- **Scalability :** The indexing system is designed to scale with the dataset, ensuring that the performance remains optimal even as the volume of anime/manga descriptions grows.

By structuring the indexing component in this manner, the system can efficiently manage and retrieve information related to terms and their frequencies within the anime/manga dataset.

4. Search and Ranking

Types of Search:

- **Query Search :** Involves searching based on user-provided queries.
- **Recommendation Search :** Provides recommendations based on the user's watch/read history.

Term Frequency Calculation:

- Term frequency is calculated using the formula: 1 + log2(tf).

Inverse Document Frequency Calculation:

- Inverse Document Frequency (IDF) is calculated as: 1 + log2(total_Doc/Doc_freq).

Weight Index:

- The weight index for each term is determined by multiplying the term frequency (tf) and inverse document frequency (idf): tf * idf.

Recommendation Search Workflow:

- **User Watch/Read History :** The system gathers the watch/read history of the user.
- **Weight Index Calculation :** Calculates the weight index of each document that the user has watched/read.

- **Query Weight Index :** Obtains the query weight index by summing the weight indices of the user's watch/read history.
- **Cosine Similarity :** Calculates cosine similarity to determine the similarity between each document's weight index and the query weight index.

Cosine Similarity:

- Measures the cosine of the angle between two vectors, providing a similarity score.

Search Optimization:

- Utilizes weighted indices, term frequency, and inverse document frequency for accurate and relevant search results.
- Cosine similarity ensures efficient comparison and ranking of documents based on their relevance.

Result Presentation:

- Presents search results in a ranked order, with higher cosine similarity scores indicating higher relevance.

5. Spell Checking

- **The correct_spelling function :** is a spell checker designed to identify and correct misspelled words in a given query.
- **Library Used:** The function utilizes the enchant library, specifically the English (U.S.) dictionary, to check word spelling.
- **Input:** Accepts a query (text) string as input.
- **Processing:**
  - Splits the input query into individual words.
  - Checks if each word is misspelled using the enchant dictionary.
  - If misspelled, suggests a correction using the first suggestion from the dictionary.
  - Assembles corrected words into a new query string.
- **Output:** Returns the corrected query.

6. Feedback

Feedback IDs Extraction:

- The system accepts a list of feedback IDs from the user, obtained from the JSON payload using request.json.get('feedbackIDs', []).

Feedback Function:

- The feedback() function processes received feedback IDs, updating the dataset by incrementing feedback count, recalculating evaluation based on rating, and computing normalized evaluation. The updated dataset is saved to 'main dataset/main.csv', and feedback IDs are printed for confirmation.

Normalized Evaluation Map:

- The function retrieves the normalized evaluation map using the getNormalizedEvaluation function from the 'preprocessingAndIndexing' module.

By incorporating this feedback component, the system allows users to provide feedback on specific IDs, updating the dataset for continuous improvement in recommendations or search results.

7. Assessment: Information Retrieval (IR) Evaluation

Precision-Recall Curve Generation:

- The PRcurve module generates Precision-Recall curves to evaluate the performance of the Information Retrieval (IR) system.

PRcurve Function:

- Takes parameters including total_docs, relevant_docs_id, and top_10 to calculate precision and recall at each rank in the top 10. Interpolation is applied, and the resulting curve is plotted using Matplotlib.

Workflow:

- **Relevance Annotation:** Determines the relevance of each document in the top 10 based on provided relevant document IDs.
- **Precision and Recall Calculation:** Calculates precision and recall at each rank, providing insights into system performance.
- **Interpolation:** Interpolates precision values at 11 recall levels, producing a smooth Precision-Recall curve.
- **Curve Plotting:** Plots the Precision-Recall curve for visualization.

Result Interpretation:

- Precision and recall values break down the performance at each rank in the top 10.
- Interpolated precision values offer a comprehensive view of the system's behavior across different recall levels, aiding in evaluation and comparison of retrieval strategies.

8. User Interface (UI) Component

The UI component of the Anime/Manga Recommendation System employs Flask, HTML, CSS, and JavaScript for an interactive user experience. Key elements include a spell checker, Flask routes for processing user input and feedback, HTML templates, and JavaScript for user interactions.

Components:

- **Spell Checker :** The correct_spelling function corrects misspelled words using the enchant library.
- **Flask Routes :**
- **/** : Renders the main HTML page.
- **/process_user_input :** Processes user queries and recommendations.
- **/process_feedback_input :** Handles user feedback.
- **HTML Templates :** The main template (index.html) defines the UI structure, including forms, loading indicators, and result containers.
- **JavaScript Interaction :** JavaScript functions handle user interactions like user type selection, entering IDs, and form submissions.

User Interaction Workflow:

- **User Type Selection :** Users choose existing or guest status.
- **User ID and Options :** For existing users, ID entry and dynamic options for query or recommendations are provided.
- **Query and Page Input :** Users enter queries or page numbers.
- **Search Execution :** Clicking search triggers Flask to process input and display results.
- **Feedback Submission :** Users can submit feedback, updating the system dataset.

Loading and Feedback:

- Loading indicators signal ongoing processes.
- Feedback prompts are presented after search results for user contribution.

Contribution

Kanishaka Pranjal (S20210010108)(kanishaka.p21@iiits.in):

- Project Management
- Dataset Collection & Management
- Search and Ranking (done by both)
- User Feedback
- Spell Checking
- User Interface (UI) Development

Swastik Mukati (S20210010221)(swastik.m21@iiits.in):

- PreProcessing
- Indexing
- Recommendation and Similarity algorithm
- Search and Ranking (done by both)
- Assessment: Information Retrieval (IR) Evaluation
- Testing and Debugging